ARTIFICIAL INTELLIGENCE

(CSC 462)

LAB # 5

NAME: MUAAZ BIN MUKHTAR

REG NO: FA21-BSE-045

CLASS & SECTION: BSSE-5A

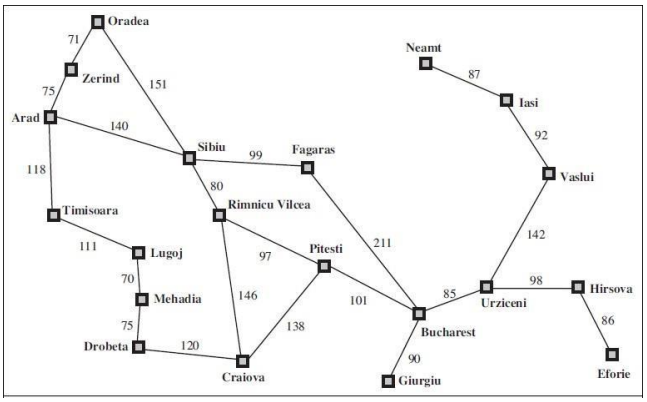
SUBMITTED TO: SIR WAQAS ALI

DATE SUBMITTED: 02-10-2023

Department of Computer Science

Lab Task 1:

Imagine going from Arad to Bucharest in the following map. Your goal is to minimize the distance mentioned in the map during your travel. Implement a iterative deepening search to find the corresponding path.



Code:

```
graph = {
    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},
    'Zerind': {'Arad': 75, 'Oradea': 71},
    'Timisoara': {'Arad': 118, 'Lugoj': 111},
    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},
    'Oradea': {'Zerind': 71, 'Sibiu': 151},
    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
    'Rimnicu Vilcea': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},
    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},
    'Drobeta': {'Mehadia': 75, 'Craiova': 120},
    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},
    'Pitesti': {'Craiova': 138, 'Rimnicu Vilcea': 97, 'Bucharest': 101},
    'Bucharest': {'Fagaras': 211, 'Pitesti': 101}
}

def iterative_deepening_search(start, goal, graph):
    depth = 0
    while True:
        result = depth_limited_search(start, goal, graph, depth)
        if result is not None:
            return result
        depth += 1

def depth_limited_search(node, goal, graph, depth):
    if depth == 0 and node == goal:
        return [node]
    elif depth > 0:
        for neighbor, _ in graph[node].items():
            result = depth_limited_search(neighbor, goal, graph, depth-1)
            if result is not None:
                return [node] + result
```

```
start_node = 'Arad'
goal_node = 'Bucharest'

path = iterative_deepening_search(start_node, goal_node, graph)

if path is not None:
    print("Path found:", ' -> '.join(path))
else:
    print("No path found.")
```

Output:

```
C:\Users\FA21-BSE-009\PycharmProjects\lab6\venv\Scripts\pyth
('Path found:', 'Arad -> Sibiu -> Fagaras -> Bucharest')

Process finished with exit code 0
```

Lab Task 2:

Generate a list of possible words from a character matrix

Given a 8 × 8 boggle board, find a list of all possible words that can be formed by a sequence of adjacent characters on the board. We are allowed to search a word in all eight possible directions, i.e., North, West, South, East, North-East, North-West, South-East, South-West, but a word should not have multiple instances of the same cell.

Consider the following the traditional 4 × 4 boggle board. If the input dictionary is [START, NOTE, SAND, STONED], the valid words are [NOTE, SAND, STONED]. With iterative deepening, create words of length 5, 6, 7 and 8 through each iteration.

M	S	E	F
R	A	T	D
L	O	N	E
K	A	F	B

Figure 15 - 4x4 Boggle Board

Code:

```
def is_valid(x, y, visited):
    return 0 <= x < 4 and 0 <= y < 4 and not visited[x][y]

def iddfs(board, words):
    def dfs(x, y, visited, path):
        visited[x][y] = True
        path += board[x][y]

        if len(path) >= 5 and path in words_set :
            result.add(path)

        if len(path) < 8:
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    if dx == 0 and dy == 0:
                        continue
                    nx, ny = x + dx, y + dy
                    if is_valid(nx, ny, visited):
                        dfs(nx, ny, visited, path)

        visited[x][y] = False

    words_set = set(words)
    result = set()

    for i in range(4):
        for j in range(4):
            visited = [[False for _ in range(4)] for _ in range(4)]
            dfs(i, j, visited, "")

    return result
```